



Tecnológico de Monterrey

**Instituto Tecnológico de Estudios Superiores de
Monterrey**

TE3001B.101

Fundamentación de Robótica

Challenge 1.

Gpo 101

Profesor:

Rigoberto Cerino Jiménez

Alumnos:

Daniela Berenice Hernández de Vicente	A01735346
Alejandro Armenta Arellano	A01734879
Dana Marian Rivera Oropeza	A00830027

Fecha: 20 de Febrero del 2023

Resumen

En este reporte se podrá observar una descripción de lo que es ROS, cómo funciona y cómo está organizado, todo esto con tal de comprender más allá de simplemente resolver el challenge impuesto por Manchester Robotics.

Aunado a estas descripciones y definiciones, se puede encontrar la solución paso por paso al challenge de esta semana.

Objetivos

En este primer challenge se espera que nosotros podamos resolver la actividad de la siguiente forma:

- Creando dos nodos.
- El primero de ellos debe actuar como un generador de señales, por lo que deberá generar una señal senoidal.
- El segundo nodo debe actuar como un proceso, el cual debe tomar la señal generada por el nodo anterior y modificarla, generando así una señal procesada.
- Ambas señales deben ser impresas con la instrucción `rqt_plot`.
- En diferentes terminales se debe mostrar la información generada por las señales.
- Debe ser creado un launch file para lanzar ambos nodos, terminales y la instrucción `rqt_plot` al mismo tiempo.

Introducción

Semanalmente se nos presentará un reto por Manchester Robotics, los cuales serán resueltos con la ayuda de programas especializados como ROS y ciertos lenguajes de programación como Python.

De igual manera cada semana contaremos con una clase base con respecto a lo que se manejara durante el challenge semanal, así como una sesión de preguntas y respuestas donde se buscarán resolver dudas con respecto al mismo challenge de la semana.

En esta primera semana se nos introdujo de una forma amable a ROS, enseñándonos que se trata de una estructura open-source que ayuda a los investigadores y desarrolladores a construir y reutilizar código en aplicaciones de robótica.

Consta de una arquitectura particular, la cual busca la comunicación e interacción entre cada uno de sus componentes.

- ROS Master:
 - Estructura principal que le permite a ROS trabajar.
 - De igual manera gestiona los procesos que permiten la comunicación entre los diferentes componentes en la red.
 - Permite la comunicación entre los diferentes nodos, los cuales se pueden encontrar en diferentes computadoras o robots mediante una arquitectura Cliente-Servidor.
- ROS Nodes:
 - Pieza del software que actúa como un elemento en la red.
 - Ejecuta parte del código, puede ser programado en C++ o Python.
- Tópicos:

- Pueden ser nombrados buses a través de los cuales los nodos pueden intercambiar mensajes.
- Están destinados a la comunicación de transmisión unidireccional.
- Pueden haber múltiples varios publishers y subscribers con respecto a un tópico.
- Messages:
 - Los nodos se comunican entre sí mediante la publicación de mensajes en los tópicos.
 - Un mensaje es simplemente una estructura de datos, los cuales comprenden campos escritos.
 - Muchos tipos de datos son soportados, como los enteros, los puntos flotantes o booleanos.
 - Los mensajes pueden incluir estructuras anidadas y arreglos.

De igual manera conocimos la organización que puede llegar a manejar el sistema operativo ROS.

- ROS Workspace:
 - Los proyectos en ROS están organizados en espacios trabajo, los cuales son una colección de códigos agrupados denominados **paquetes**.
 - Son un set de directorios o folders donde se almacenan piezas relacionadas de código en ROS.
 - El nombre oficial para los espacios de trabajo en ROS **catkin workspace**.
- Catkin Workspace:
 - src: Denominados “**Source Space**”, contiene el código fuente. En este espacio es donde los usuarios pueden crear, copiar o editar el código fuente para construir diferentes paquetes.
 - build: Este espacio está reservado para el archivo CMake el cual se encarga de construir los paquetes dentro de la carpeta src. En este archivo se concentra la información cache y otros archivos intermediarios los cuales se usan durante la construcción de los paquetes, es IMPORTANTE NO TOCARLO.
 - devel: El espacio de desarrollo, es donde se construyen los objetivos y los ejecutables son colocados antes de ser instalados, es IMPORTANTE NO TOCARLO.
- ROS Packages:
 - Para mantener todo organizado, ROS utiliza el concepto de paquetes.
 - Son una forma de organizar el código relacionado unos con otros.
 - Cada paquete requiere de diferentes atributos para ser compilado.
 - Estos atributos usualmente son dependencias relacionadas con otros paquetes, librerías externas o mensajes personalizados, servicios y acciones.
 - La herramienta de compilación preferida se conoce como **catkin** y usa dos archivos separados, **package.xml** y **CMakeLists.txt**.
 - Las instrucciones para compilar deben ser asignadas en CMake y Package files.

De manera resumida podemos decir lo siguiente:

- Packages: Archivos que pueden ser exportados entre proyectos.
- Los archivos de configuración son utilizados para establecer dependencias de código.
- Los archivos y folders extras, son archivos dedicados para tareas específicas. En este caso el Launch file nos permite ejecutar varios nodos al mismo tiempo.
- Nodes: Contiene código que ejecutaremos dentro de cada nodo.

Solución del problema

Presentar la metodología realizada para lograr los objetivos del reto, así como la descripción de los elementos, funciones y seguimiento del código de programación implementado.

Primero creamos el paquete courseworks.

```
berenice@berenice-IdeaPad-3-15IIL05: ~/catkin_ws/src
Archivo Editar Ver Buscar Terminal Ayuda
berenice@berenice-IdeaPad-3-15IIL05:~$ cd ^C
berenice@berenice-IdeaPad-3-15IIL05:~$ cd ~
berenice@berenice-IdeaPad-3-15IIL05:~$ cd ~/catkin_ws/src
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src$ catkin_create_pkg courseworks std_msgs rospy
Created file courseworks/CMakeLists.txt
Created file courseworks/package.xml
Created folder courseworks/src
Successfully created files in /home/berenice/catkin_ws/src/courseworks. Please adjust the values in package.xml.
```

Posteriormente creamos la carpeta scripts donde se almacenarán los archivos fuente de nuestro código en python. De igual manera creamos los dos nodos y se les da permisos para desarrollarse en ubuntu.

```
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws$ cd src
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src$ cd courseworks
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks$ mkdir scripts
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks$ cd scripts
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts$ touch signal_generator.py
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts$ touch process.py
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts$ sudo chmod +x signal_generator.py
[sudo] contraseña para berenice:
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts$ sudo chmod +x process.py
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts$
```

Después creamos el archivo launch, el cual será el encargado de ejecutar ambos nodos al mismo tiempo.

```
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/scripts$ cd ..
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks$ mkdir launch
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks$ cd launch
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/launch$ touch activity1.launch
berenice@berenice-IdeaPad-3-15IIL05:~/catkin_ws/src/courseworks/launch$
```

Donde en su interior tiene el siguiente código:

```
he > alex > catkin_ws > src > courseworks > launch > challenge1.launch
1 <?xml version="1.0" ?>
2 <launch>
3
4 <node name="signal_generator" pkg="courseworks" type="signal_generator.py"
5 <node name="process" pkg="courseworks" type="process.py" output="screen"
6
7 </launch>
8
```

Aunado a eso modificamos el archivo de CMakeLists.txt para que ambos nodos puedan interactuar con el programa.

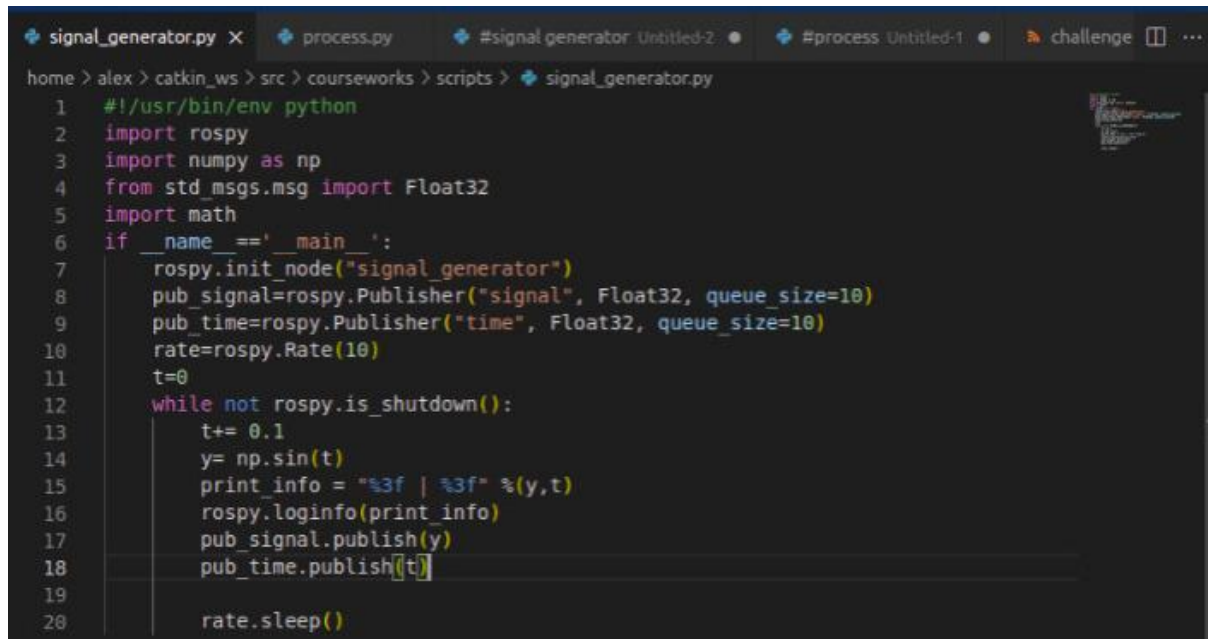
```

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
catkin_install_python(PROGRAMS
    scripts/signal_generator scripts/process
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

```

Posteriormente procedemos a generar los códigos en python para ambos nodos.

Nodo Signal_generator:



```

1  #!/usr/bin/env python
2  import rospy
3  import numpy as np
4  from std_msgs.msg import Float32
5  import math
6  if __name__ == '__main__':
7      rospy.init_node("signal_generator")
8      pub_signal=rospy.Publisher("signal", Float32, queue_size=10)
9      pub_time=rospy.Publisher("time", Float32, queue_size=10)
10     rate=rospy.Rate(10)
11     t=0
12     while not rospy.is_shutdown():
13         t+= 0.1
14         y= np.sin(t)
15         print_info = "%3f | %3f" %(y,t)
16         rospy.loginfo(print_info)
17         pub_signal.publish(y)
18         pub_time.publish(t)
19
20     rate.sleep()

```

En este código se genera y manda una señal en base a la siguiente función trigonométrica: $y = f(t) = \sin(t)$. Se manda a publicar un mensaje estándar de ROS en un tópico denominado **signal**. De igual manera se publica el tiempo en otro tópico denominado tiempo, usando el mismo tipo de mensaje. Se utiliza una tasa de 10 Hz para este nodo.

Nodo Process:

```
home > alex > catkin_ws > src > courseworks > scripts > process.py
1  #!/usr/bin/env python
2  import rospy
3  import numpy as np
4  from std_msgs.msg import Float32
5
6  original_signal = 0
7  original_time = 0
8
9  def callbacksignal(msgsignal):
10     global original_signal
11     original_signal = msgsignal.data
12
13  def callbacktime(msgtime):
14     global original_time
15     original_time = msgtime.data
16
17  if __name__ == '__main__':
18     rospy.init_node('process')
19     rospy.Subscriber("signal", Float32, callbacksignal)
20     rospy.Subscriber("time", Float32, callbacktime)
21
22     pub_signal = rospy.Publisher("process", Float32, queue_size = 10)
23     pub_time = rospy.Publisher("time_2", Float32, queue_size = 10)
24     rate = rospy.Rate(10) #10 Hz
25
26     while not rospy.is_shutdown():
27         t=original_time
28
29         y=((original_signal * np.cos(5)) + np.cos(t) * np.sin(5))* -np.cos(t)
30         print_info = "%3f | %3f" %(y,t)
31         rospy.loginfo(print_info)
32         pub_signal.publish(y)
33         pub_time.publish(t)
34
35     rate.sleep()
```

En este código se crea un nuevo nodo que se suscribe a los tópicos de tiempo y señal. En este se procesa la señal anteriormente creada, generando un offset que siempre se mantenga positivo igual a 0. De igual manera se reduce la amplitud de la señal recibida a la mitad. Se añade un cambio de fase a la señal recibida (como parámetros de usuario o variable) a la señal original. De igual forma se utiliza una tasa de 10 Hz para este nodo, y se imprimen los resultados en la terminal como se muestra en el apartado de resultados.

Es importante conocer que para lanzar ambos nodos es necesario tener corriendo ROS con los siguientes comandos:

```
alex@alex-Lenovo-Y720-15IKB:~/catkin_ws$ roslaunch courseworks challenge1.launch
... logging to /home/alex/.ros/log/561cc2e4-b16d-11ed-a756-60f677daca7d/roslaunch
h-alex-Lenovo-Y720-15IKB-20243.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://alex-Lenovo-Y720-15IKB:43623/

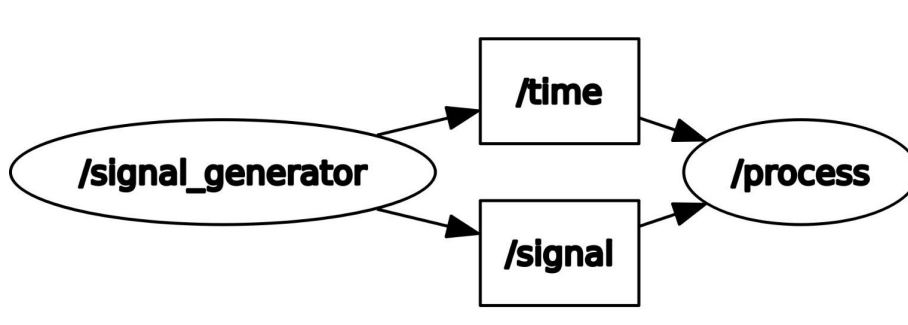
SUMMARY
=====

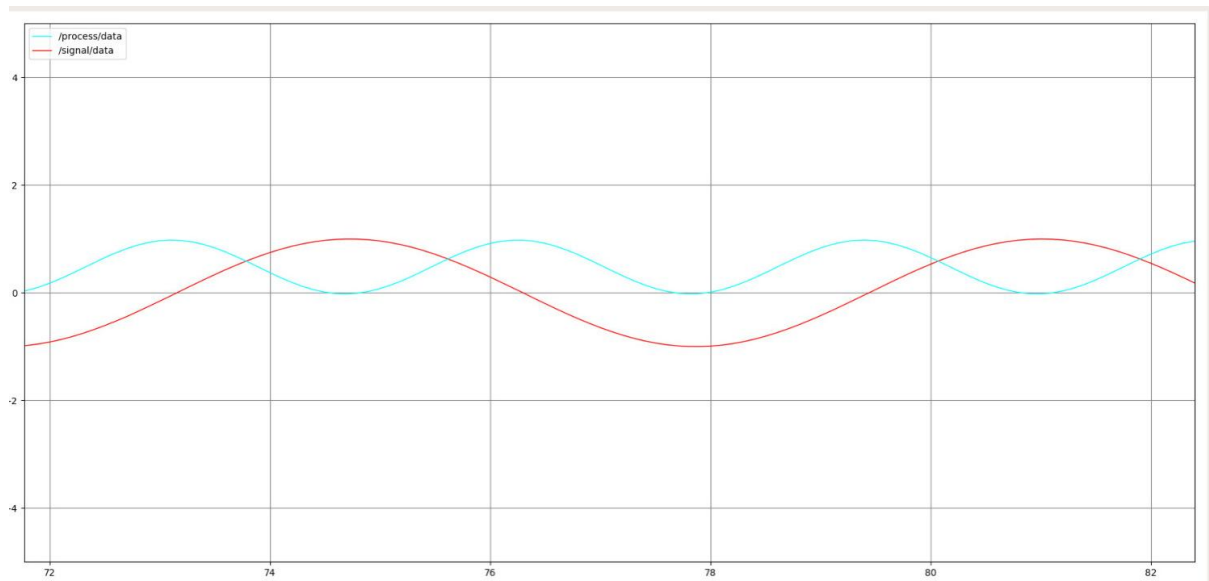
PARAMETERS
* /rostdistro: melodic
* /rosversion: 1.14.13

NODES
/
  process (courseworks/process.py)
  signal_generator (courseworks/signal_generator.py)
```

Resultados

En este apartado se muestra la relación que existe entre los canales de comunicación entre los nodos para la transmisión de la señal senoidal y su procesamiento, después se muestra la gráfica de ambas señales dejando ver la diferencia entre la señal procesada y la original, por último están las impresiones de los datos que se comunican en los canales, el tiempo recuperado y el valor de seno con respecto al tiempo que se está considerando.





Terminal			Terminal		
Archivo	Editar	Ver	Archivo	Editar	Ver
Buscar	Terminal	Ayuda	Buscar	Terminal	Ayuda
INFO	[1676933689.616820]:	-0.693525 11.800000	INFO	[1676933689.619020]:	0.639432 11.800000
INFO	[1676933689.716919]:	-0.618137 11.900000	INFO	[1676933689.719301]:	0.639432 11.800000
INFO	[1676933689.816933]:	-0.536573 12.000000	INFO	[1676933689.819191]:	0.712169 11.900000
INFO	[1676933689.916763]:	-0.449647 12.100000	INFO	[1676933689.919159]:	0.790472 12.000000
INFO	[1676933690.016975]:	-0.358229 12.200000	INFO	[1676933690.019236]:	0.930739 12.200000
INFO	[1676933690.116785]:	-0.263232 12.300000	INFO	[1676933690.119212]:	0.990512 12.300000
INFO	[1676933690.216941]:	-0.165604 12.400000	INFO	[1676933690.219272]:	0.978953 12.400000
INFO	[1676933690.316810]:	-0.066322 12.500000	INFO	[1676933690.319052]:	0.973478 12.500000
INFO	[1676933690.416807]:	0.033623 12.600000	INFO	[1676933690.419279]:	0.948308 12.600000
INFO	[1676933690.516866]:	0.133232 12.700000	INFO	[1676933690.519434]:	0.948308 12.600000
INFO	[1676933690.616949]:	0.231510 12.800000	INFO	[1676933690.619156]:	0.843642 12.800000
INFO	[1676933690.716815]:	0.327474 12.900000	INFO	[1676933690.719404]:	0.768320 12.900000
INFO	[1676933690.816868]:	0.420167 13.000000	INFO	[1676933690.819071]:	0.681481 13.000000
INFO	[1676933690.916843]:	0.508661 13.100000	INFO	[1676933690.919365]:	0.586588 13.100000
INFO	[1676933691.016862]:	0.592074 13.200000	INFO	[1676933691.019059]:	0.487425 13.200000
INFO	[1676933691.116747]:	0.669570 13.300000	INFO	[1676933691.119120]:	0.387944 13.300000
INFO	[1676933691.216898]:	0.740376 13.400000	INFO	[1676933691.219016]:	0.292112 13.400000
INFO	[1676933691.316877]:	0.803784 13.500000	INFO	[1676933691.319257]:	0.203749 13.500000
INFO	[1676933691.416972]:	0.859162 13.600000	INFO	[1676933691.419003]:	0.126377 13.600000
INFO	[1676933691.516828]:	0.905955 13.700000	INFO	[1676933691.519233]:	0.063082 13.700000
INFO	[1676933691.616857]:	0.943696 13.800000	INFO	[1676933691.619010]:	0.016387 13.800000
INFO	[1676933691.716968]:	0.972008 13.900000	INFO	[1676933691.719484]:	0.016387 13.800000
INFO	[1676933691.816901]:	0.990607 14.000000	INFO	[1676933691.819131]:	-0.020494 14.000000

Conclusiones

Aún cuando en algunos momentos parecía inalcanzable, se consiguió realizar con éxito cada uno de los objetivos antes mencionados.

Se logró obtener la señal senoidal tal y como se muestra en el apartado de resultados, de igual forma se logró indagar más en temas específicos con respecto a ROS.